Flash-Friendly File System (F2FS)

Feb 22, 2013
Joo-Young Hwang
(jooyoung.hwang@samsung.com)
S/W Dev. Team, Memory Business, Samsung Electronics Co., Ltd.

SAMSUNG
TURN ON TOMORROW

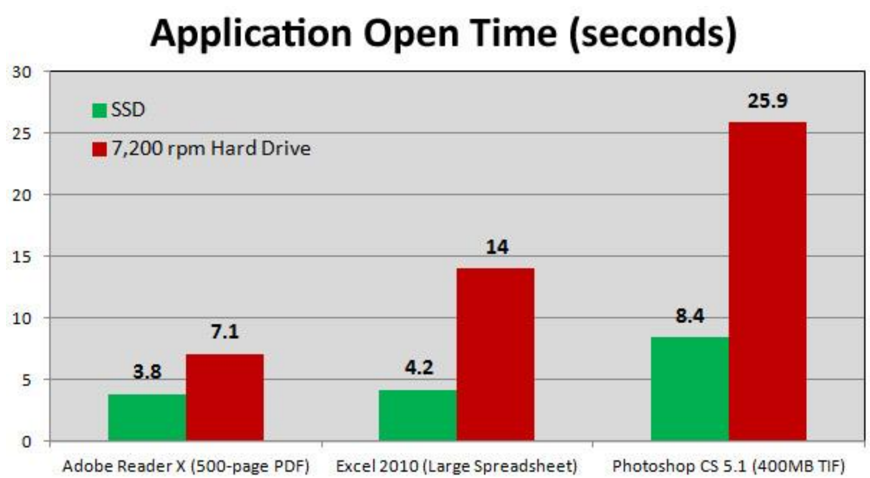
Agenda

- Introduction
- FTL Device Characteristics
- F2FS Design
- Performance Evaluation Results
- Summary

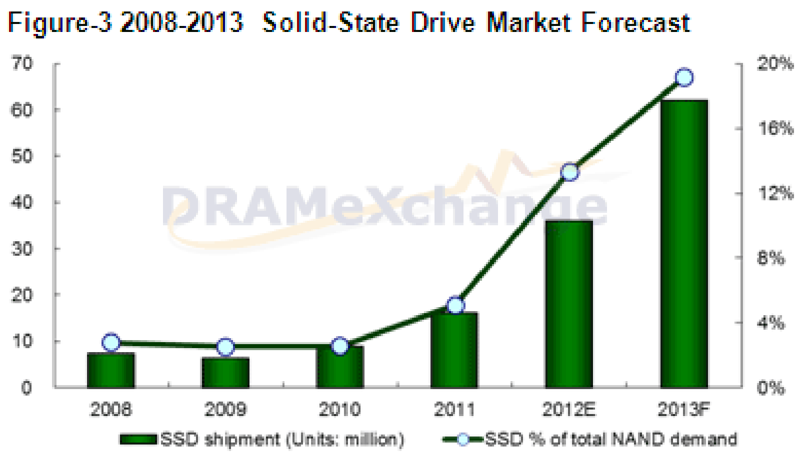
Introduction

- NAND Flash-based Storage Devices
  - SSD for PC and server systems
  - eMMC for mobile systems
  - SD card for consumer electronics

- The Rise of SSDs
  - Much faster than HDDs
  - Low power consumption



Source: March 30th, 2012 by Avram Piltch, LAPTOP Online Editorial Director



Source: DRAMeXchange, Jan., 2012

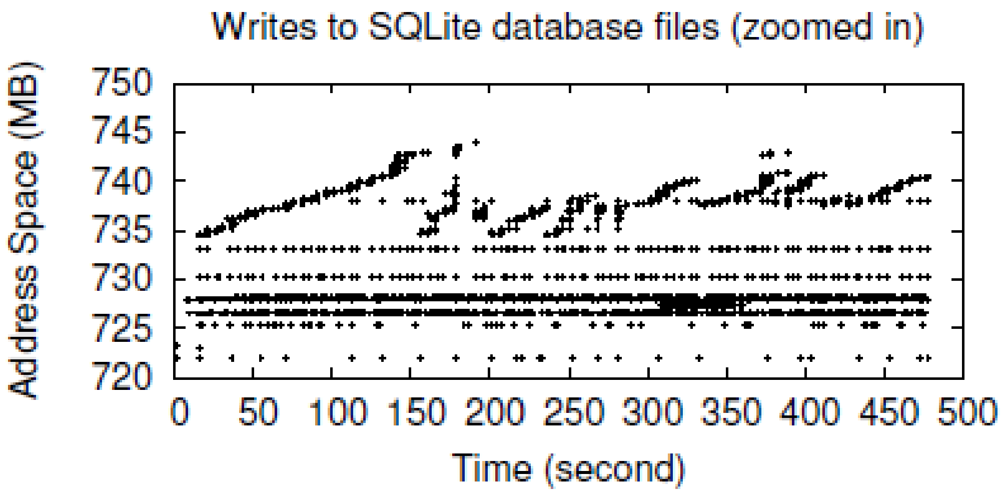
Introduction (cont'd)

- NAND Flash Memory
  - Erase-before-write
  - Sequential writes inside the erase unit
  - Limited program/erase (P/E) cycle
- Flash Translation Layer (FTL)
  - Conventional block device interface: no concern about erase-before-write
  - Address Mapping, Garbage collection, Wear Leveling
- Conventional file systems and FTL devices
  - Optimizations for HDD good for FTL?
  - How to optimize a file system for FTL device?

Storage Access Pattern in Mobile Phones

- *Sequential Write vs. Random Write*
  - Sequential write is preferred by FTL devices.

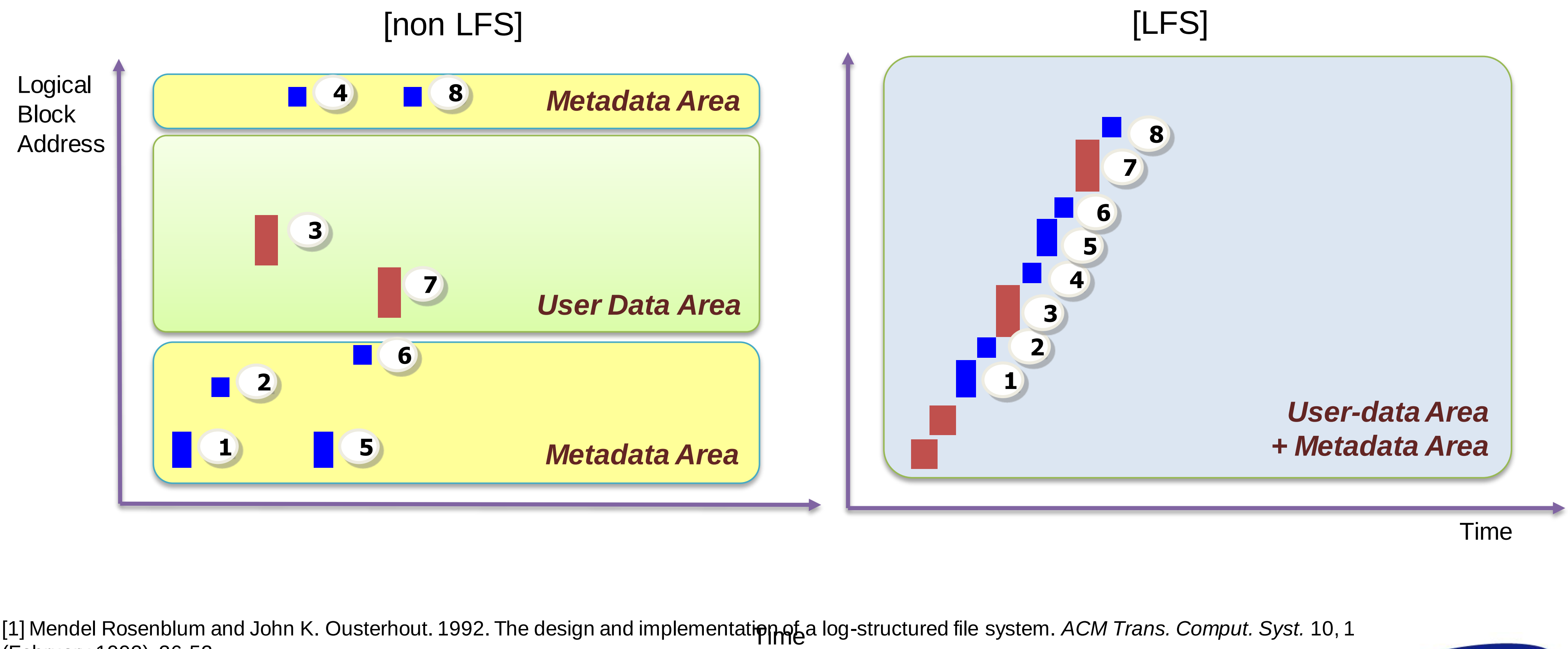
Activity	Write (MB)		Read (MB)	
	Sq	Rn	Sq	Rn
WebBench	41.3	32.2	6.8	0.5
AppInstall	123.1	5.6	0.7	0.1
Email	1.0	2.2	1.1	0.1
Maps	0.2	0.3	0	0
Facebook	2.0	3.1	0	0
RLBench	25.6	16.8	0	0
Pulse	2.6	1.0	0	0



Reference: Revisiting Storage for Smartphones, Kim et al., USENIX FAST 2012

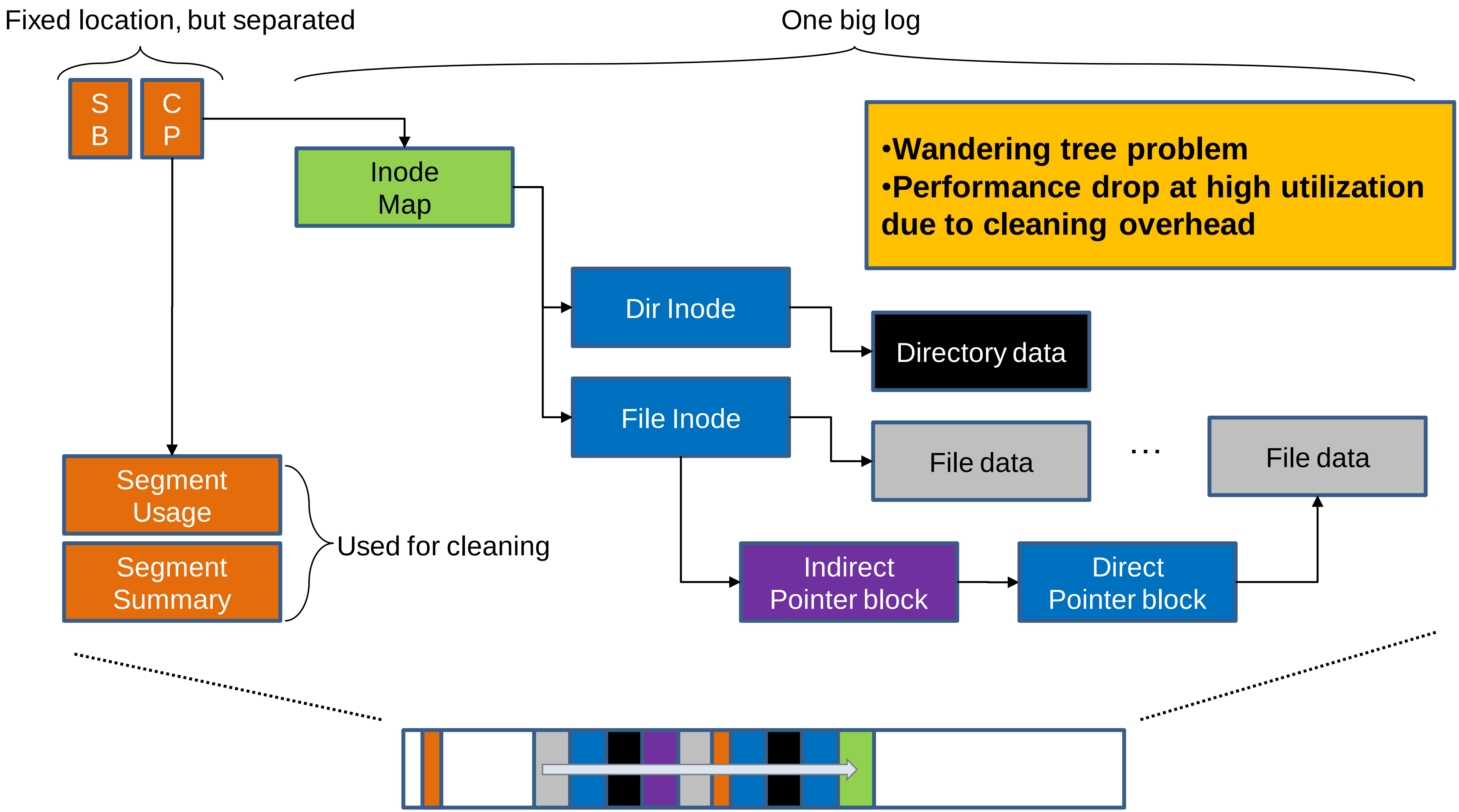
Log-Structured File System Approach for Flash Storage

- Log-structured File System (LFS)[1] fits well to FTL devices.
  - Assume the whole disk space as a big log, write data and metadata sequentially
  - Copy-on-write: recovery support is made easy.



[1] Mendel Rosenblum and John K. Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.* 10, 1 (February 1992), 26-52.

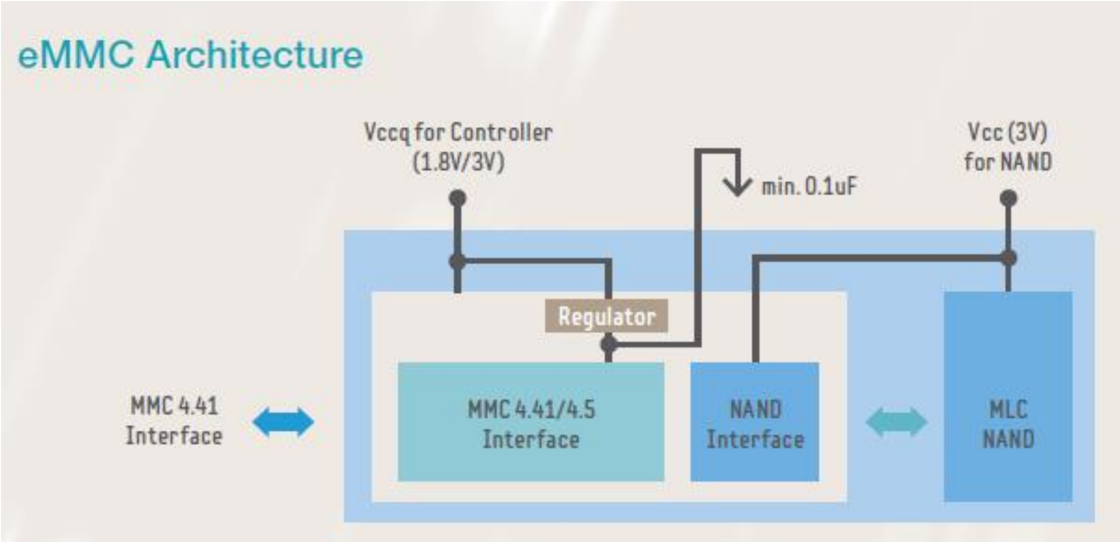
Conventional LFS



- **Wandering tree problem**
- **Performance drop at high utilization due to cleaning overhead**

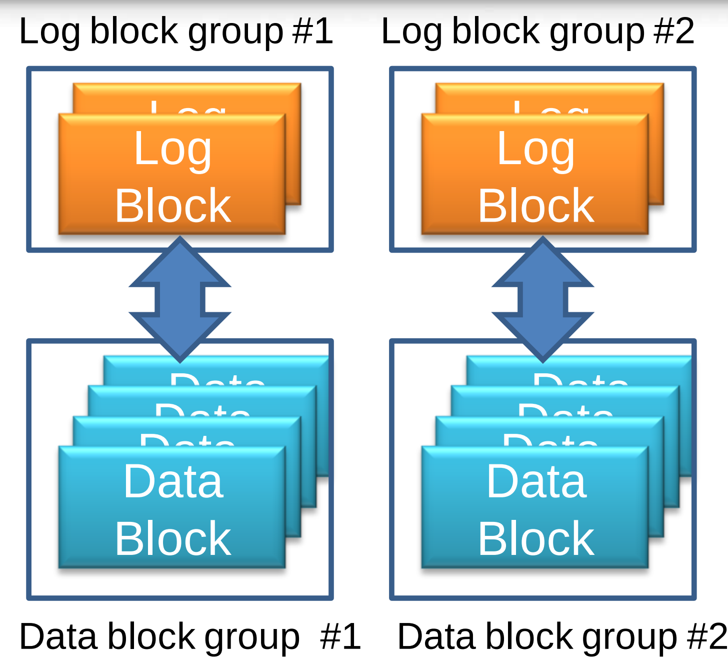
FTL Block Device

- FTL Functions
  - Address Mapping
  - Garbage Collection
  - Wear Leveling



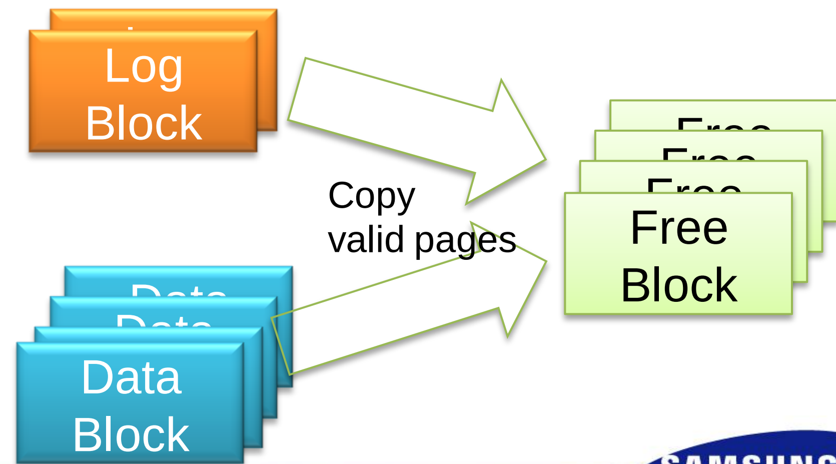
Address Mapping in FTL

- Address Mapping Methods
  - Block Mapping
  - Page Mapping
  - *Hybrid Mapping* (aka log block mapping)
    - BAST (Block Associative Sector Translation)
    - FAST (Fully Associative)
    - SAST (Set Associative)



[SAST Example – 2 log blocks per 4 data blocks]

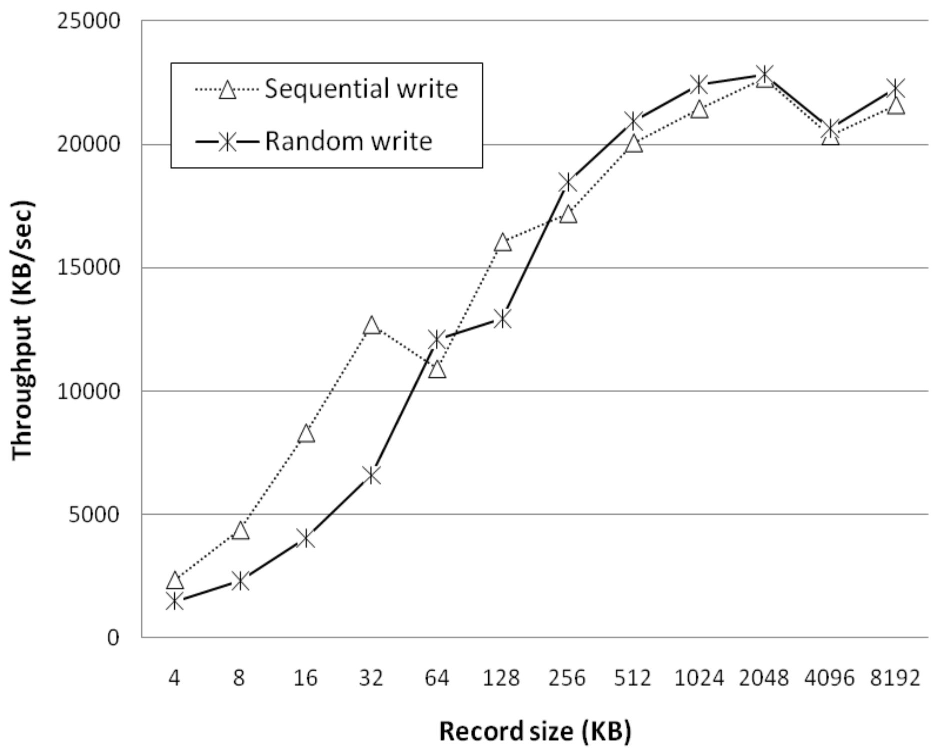
- Merge (GC in Hybrid Mapping)
  - Commit of log to data blocks
  - Merge log blocks and data block to form up-to-date data blocks
  - Merge types
    - Full merge
    - Partial merge
    - **Switch merge: most efficient!**



FTL Device Characteristics

- FTL operation unit
  - Superblock – simultaneously erasable unit
  - Superpage - simultaneously programmable unit

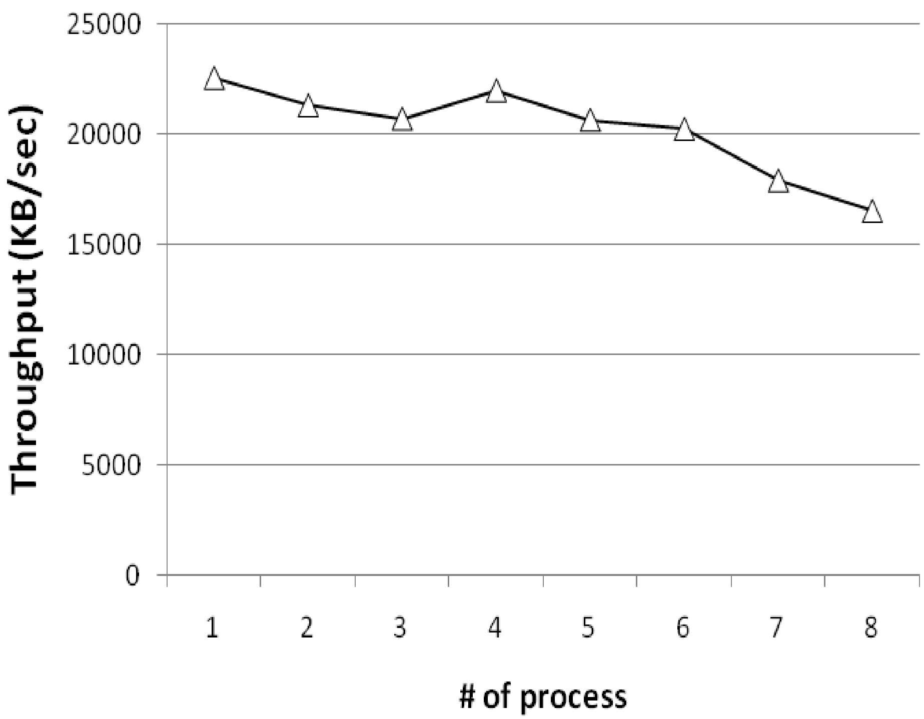
- Implications for segment size



FTL Device Characteristics (cont'd)

- FTL device may have multiple active log blocks

- Implications for multi-headed logging

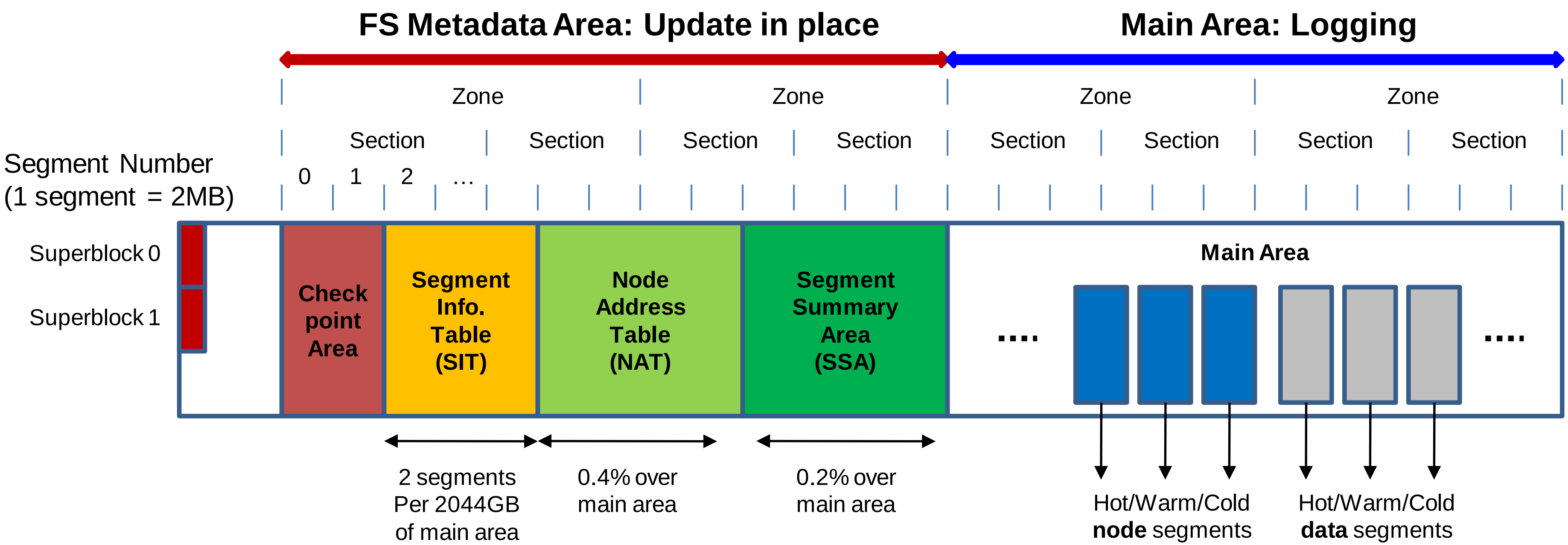


F2FS Design Overview

- FTL friendly Workload Pattern
  - To drive FTL to do switch merge in most cases
- Avoiding Metadata Update Propagation
  - Introduce indirection layer for indexing structure
- Efficient Cleaning using Multi-head Logs and Hot/Cold Data Separation
  - Write-time data separation → more chances to get binomial distribution
  - Two different victim selection policies for foreground and background cleaning
  - Automatic background cleaning
- Adaptive Write Policy for High Utilization
  - Switches write policy to threaded logging at right time
  - Graceful performance degradation at high utilization

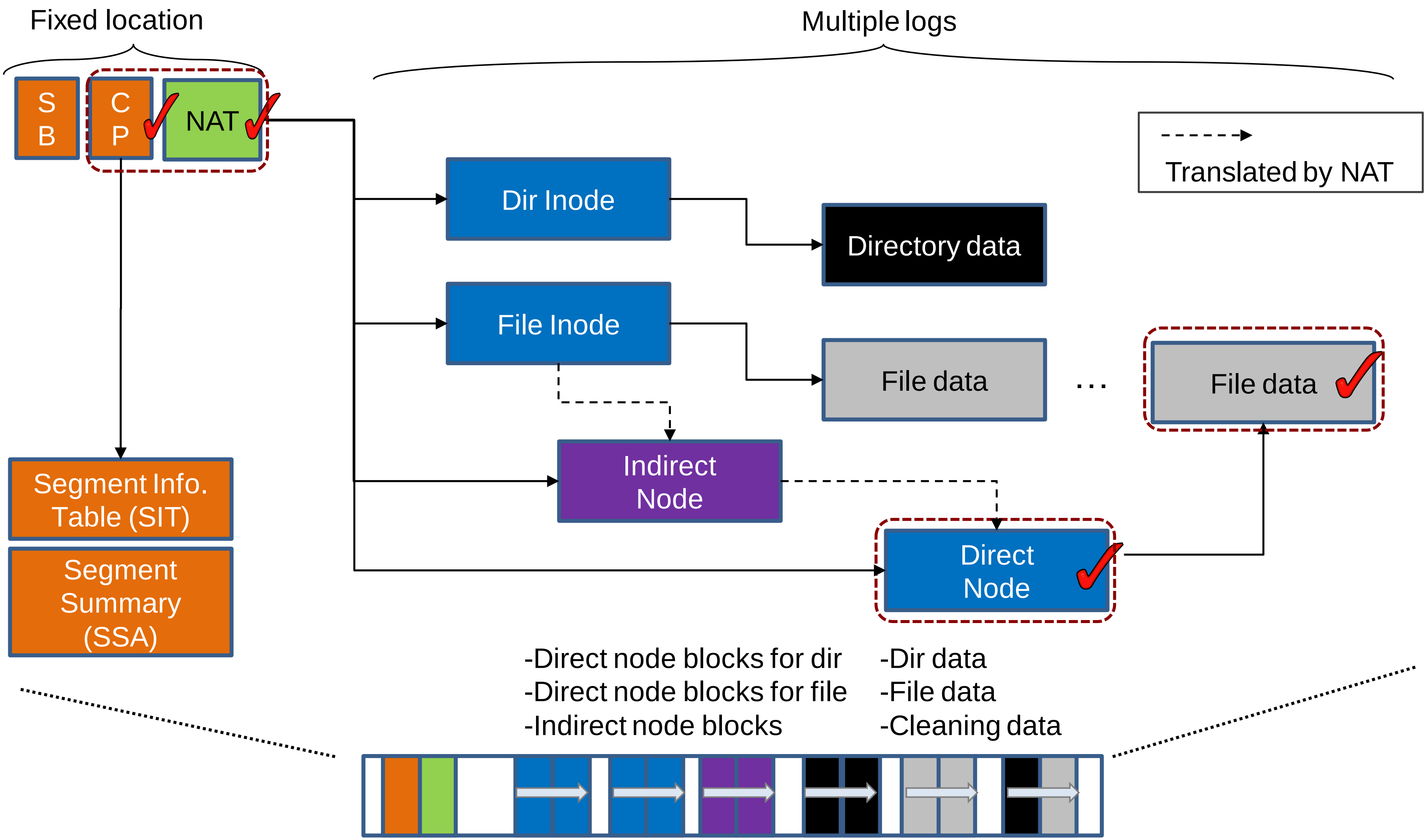
On-Disc Structure

- Start address of main area is aligned to the *zone** size
- Cleaning operation is done in a unit of *section*
  - Section is matched with FTL GC unit.
- All the FS metadata are co-located at front region.

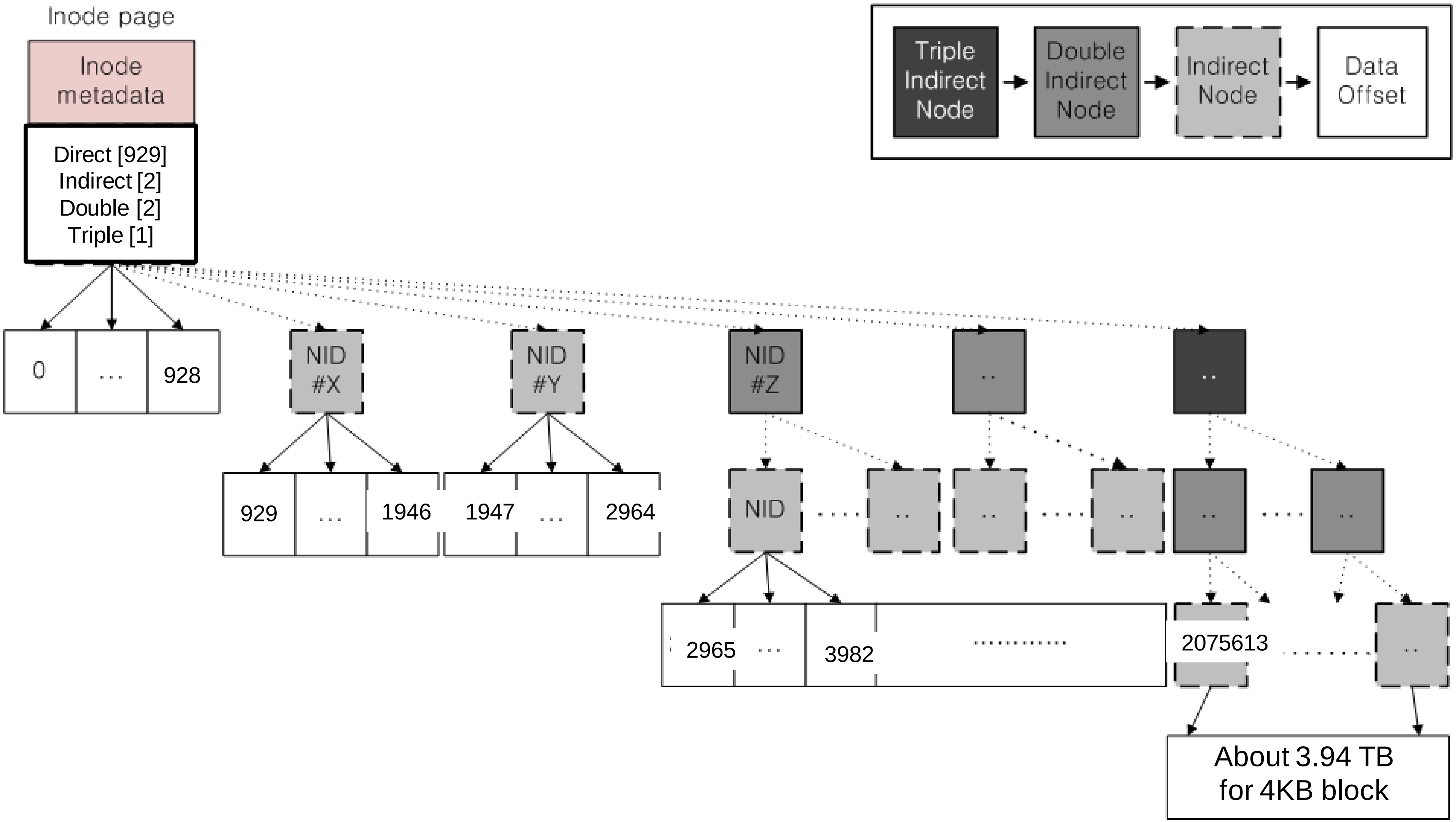


* Block size = 4KB

Addressing Wandering Tree Problem



File Indexing Structure



Cleaning

- Hot/cold data separation is a key to reducing cleaning cost.
  - Static (at data writing time)
  - Dynamic (at cleaning time)

- Hot/cold separation at data writing time based on object types
  - Cf) hot/cold separation at cleaning time requires per-block update frequency information.

Type	Update frequency	Contained Objects
Node	Hot	Directory's inode block or direct node block
	Warm	Regular file's inode block or direct node block
	Cold	Indirect node block
Data	Hot	Directory's data block
	Warm	Updated data of regular files
	Cold	Appended data of regular files, moved data by cleaning, multimedia file's data

Cleaning (cont'd)

- Dynamic hot/cold separation at background cleaning
  - Cost-benefit algorithm for background cleaning

- Automatic Background Cleaning
  - Kicked in when I/O is idle.
  - Lazy write: cleaning daemon marks page dirty, then flusher will issue I/Os later.

Adaptive Write Policy

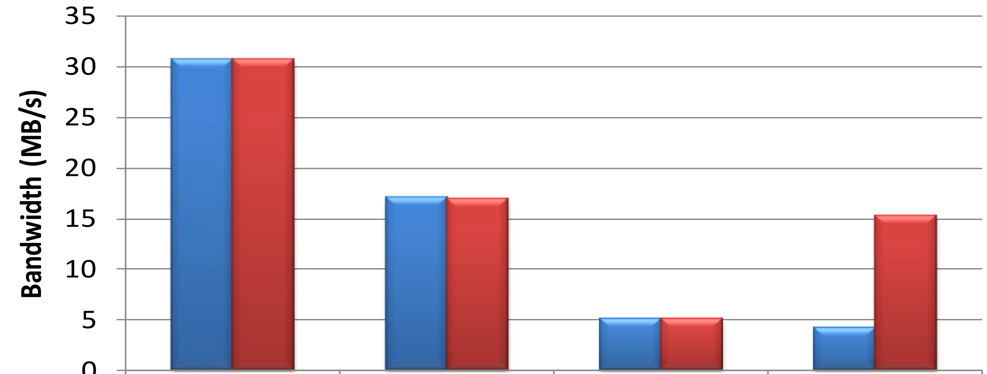
- Logging to a clean segment
  - Need cleaning operations if there is no clean segment.
  - Cleaning causes mostly random read and sequential writes.

- Threaded logging
  - When there are not enough clean segments
  - Don’t do cleaning, reuse invalidated blocks of a dirty segment
  - May cause random writes (but *in a small range*)

Performance (Panda board + eMMC)

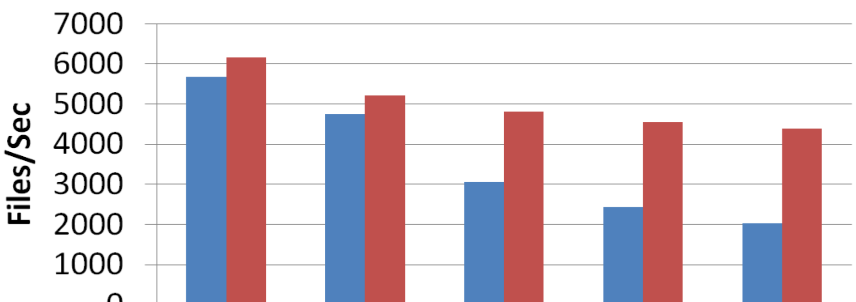
[System Specification]

CPU	ARM Cortex-A9 1.2GHz
DRAM	1GB
Storage	Samsung eMMC 64GB
Kernel	Linux 3.3
Partition Size	12 GB



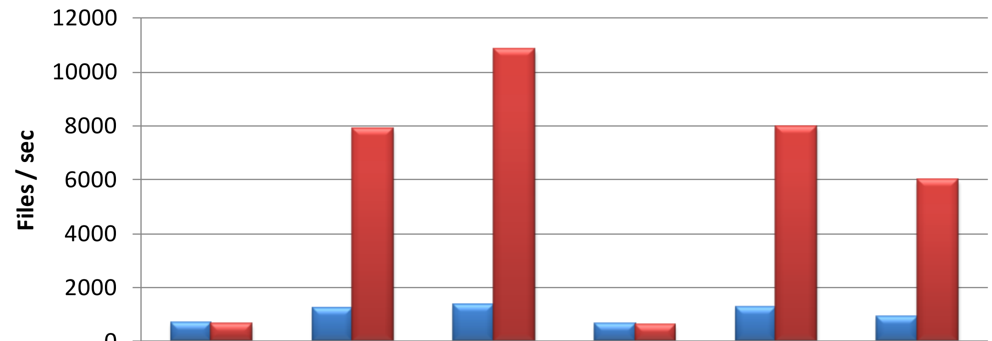
	seq. Read	seq. Write	rand. Read	rand. Write
EXT4	30.753	17.066	5.06	4.15
F2FS	30.71	16.906	5.073	15.204

[iozone]



	5000	10000	15000	20000	25000
ext4	5675.9	4741.6	3065	2439.5	2038.6
f2fs	6167.3	5207.9	4815.9	4544.4	4382.9

[fs_mark]

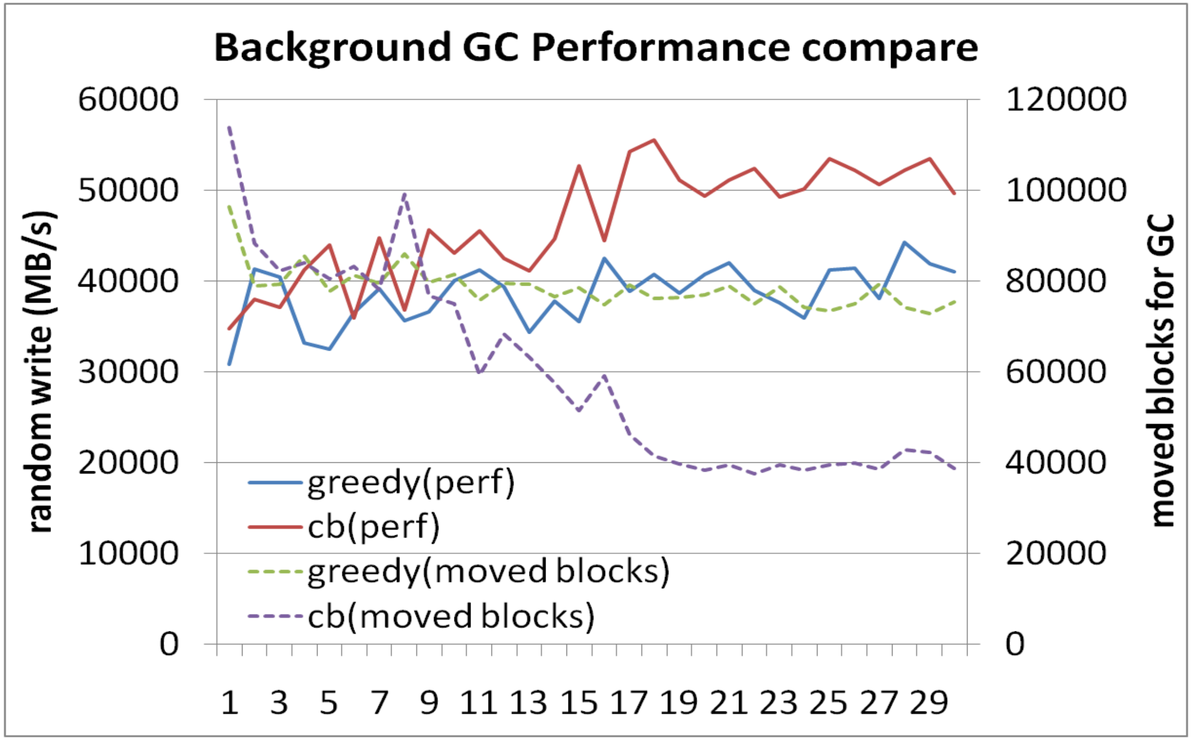


	seq.create	seq.stat	seq.delete	rand.create	rand.stat	rand.delete
EXT4	692	1238	1370	663	1250	915
F2FS	631	7871	10832	620	7962	5992

[bonnie++]

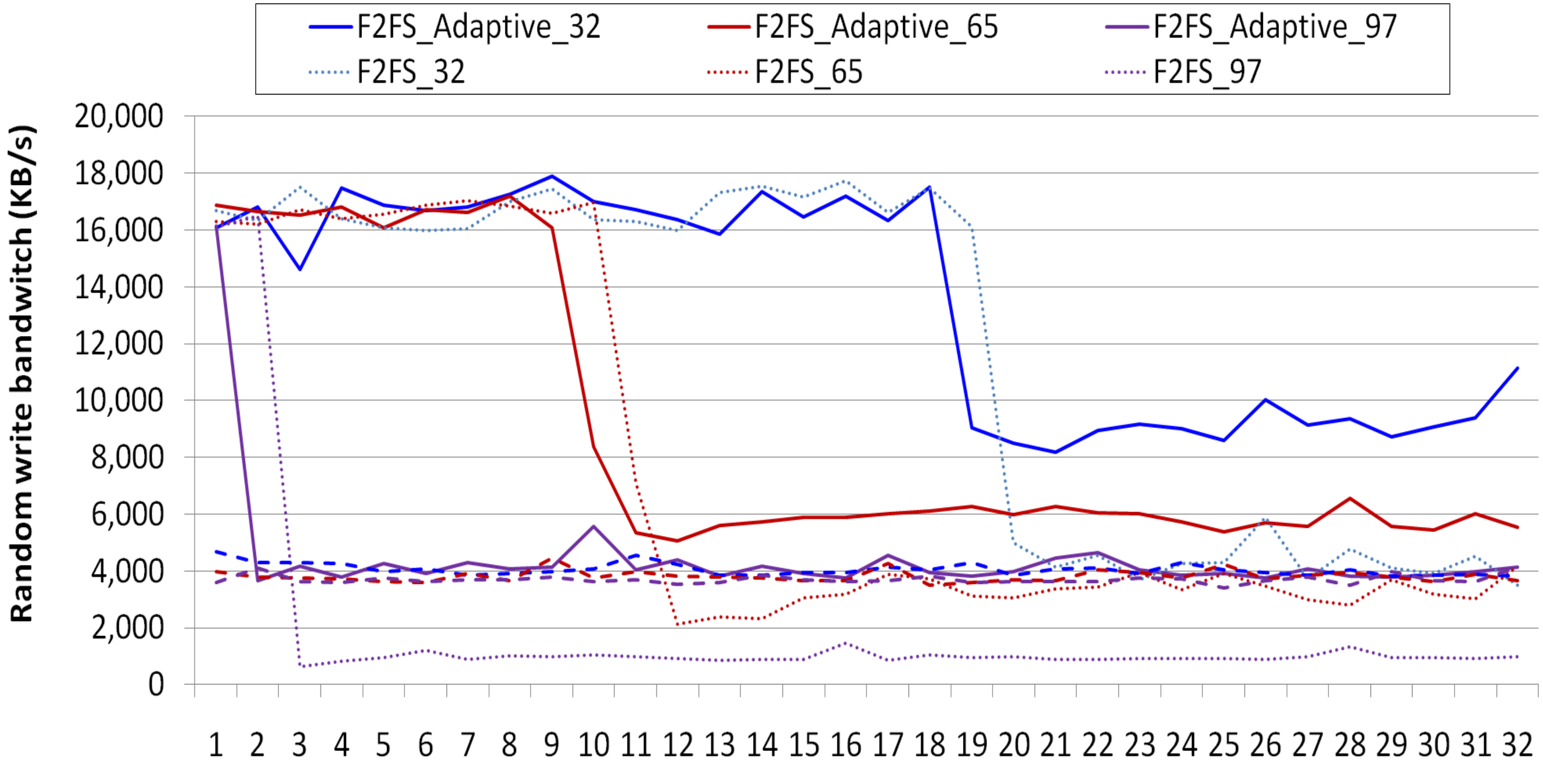
Evaluation of Cleaning Victim Selection Policies

- Setup
  - Partition size: 3.7 GB
  - Create three 1GB files, then updates 256MB randomly to each file
- Test
  - One round: updates 256MB randomly to a file
  - Iterate the round 30 times



Evaluation of Adaptive Write Policy

- Setup
  - Embedded system with eMMC 12GB partition
  - Creating 1GB files to fill up to the specified utilization.
- Test
  - Repeats Iozone random write tests on several 1GB files



Lifespan Enhancement

- Wear Acceleration Index (WAI) : total erased size / total written data

- Experiment
  - Write 12GB file sequentially.
  - Randomly update 6GB of the file.

	Ext4	F2FS
Seq Write (12GB)	1.37	1.32
Random Write (6GB)	10.70	2.29
Total	4.48	1.65

Performance on Galaxy Nexus

CPU	ARM Coretex-A9 1.2GHz
DRAM	1GB
Storage	Samsung eMMC 16GB
Kernel	3.0.8
Android ver.	Ice Cream Sandwich

< Clean >

Items		Ext4	F2FS	Improv.
Contact sync time (seconds)		431	358	20%
App install time (seconds)		459	457	0%
RLBench (seconds)		92.6	78.9	17%
IOZoneWith AppInstall (MB/s)	Write	8.9	9.9	11%
	Read	18.1	18.4	2%

< Aged >

Items		Ext4	F2FS	Improv.
Contact sync time (seconds)		437	375	17%
App install time (seconds)		362	370	-2%
RLBench (seconds)		99.4	85.1	17%
IOZone With AppInstall (MB/s)	Write	7.3	7.8	7%
	Read	16.2	18.1	12%

Summary

- Flash-Friendly File System
  - Designed for FTL block devices (not for raw NAND flash)
  - Optimized for mobile flash storages
  - Can also work for SSD

- Performance evaluation on Android Phones
  - Format /data as an F2FS volume.
  - Basic file I/O test: random write performance 3.7 times of EXT4
  - User scenario test: ~20% improvements over EXT4